DYNAMIC REGION RRT: APPLICATION TO KINODYNAMIC SYSTEMS

An Undergraduate Research Scholars Thesis

by

BEN SMITH

Submitted to the Undergraduate Research Scholars program
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Nancy Amato

May 2017

Major: Computer Science

ABSTRACT

Dynamic Region RRT: Application to Kinodynamic Systems

Ben Smith
Department of Computer Science and Engineering
Texas A&M University

Research Advisor: Dr. Nancy Amato
Department of Computer Science and Engineering
Texas A&M University

In the general motion planning problem the robot must satisfy basic constraints such as avoiding obstacles and remaining within the boundary of the environment. Kinodynamic motion planning is a type of planning where additional constraints must be satisfied. Kinodynamic planning is a more realistic planning problem as the robot must operate under constraints such as friction, gravity, velocity, and acceleration while avoiding obstacles as well. Sampling-based methods are often used to solve these types of problems. These methods generate robot configurations throughout the environment in order to eventually connect them to form a valid path from the start position to the goal. Rapidly-exploring Random Trees (RRT) are types of sampling-based methods that grow a tree from the start to goal. One important problem with these types of methods appears when planning in an environment with a narrow passage or cluttered space. In these problems it is unlikely to generate a sample in the narrow spaces and the robot does not explore these locations. Dynamic Region-biased Rapidly-exploring Random Trees (DRRRT) is a method that addresses these issues by guiding an RRT with dynamic sampling regions along an embed-

ded graph of the workspace. DRRRT is effective in general motion planning problems, but faces issues in kinodynamic problems. Oftentimes, a sample is generated near an obstacle that is valid, but is found to be unrecoverable because if the robot were to move from that state with any of the available controls it would collide with an obstacle. This often occurs in environments with narrow spaces and tight turns such as a maze.

In this work, we aim to address the problems DRRRT faces in kinodynamic problems with a series of improvements. The resulting method is compared with other motion planning techniques on two kinodynamic problems consisting of a car-like robot navigating a grid-like city and a maze, simulating narrow paths with numerous turns.

ACKNOWLEDGMENTS

There are a few people I would like to thank for the help and support they provide. Thanks to my advisor, Dr. Nancy Amato, for providing the opportunity to participate in undergraduate research. Also, thank you for continual advice and guidance on this work. I would also like to thank Read Sandström, a graduate student who I collaborated with on this work. Thanks for your help in the development of this work and your advice on working as an effective researcher. Thanks to Dr. Jory Denny for his help on developing specific aspects of this work. Finally, thanks to my family and closest friends for your constant support of my work and other endeavours.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Sampling-based motion planning is the task of determining a valid path through an environment from an initial state to a goal state by randomly selecting a state within the environment. This path is represented by a collection of states, or configurations, that are described by a set of parameters representing the location and orientation of the robot. This problem has applications in many fields, such as robotics, video games/animations, computer-aided design, and bioinformatics.

One method for solving motion planning problems is the Rapidly-exploring Random Tree (RRT) [1]. RRT is effective in single query scenarios and non-holonomic systems. Nonholonomic systems or kinodynamic systems are systems that must obey kinematic, dynamic, and force constraints [2]. In other words, in order to consider the motion from a current configuration to the next the planner must take the past configurations into consideration. The past configurations will determine the current velocities, accelerations, and momentum that are associated with the robot at that instant in time. In many environments there exists a narrow passage. These small spaces cause problems for RRTs. Dynamic Region-biased RRT (DRRRT) addresses this issue.

RRT is used as a basis for DRRRT [3], which uses an embedding graph to represent the topology and homotopy of the environment. Dynamic sampling regions are moved along the graph and sampling is biased within these regions. The embedding graph is generated by decomposing the environment and building a graph from the resulting tetrahedrons. However, this graph can often be jagged and can cause the region to be partially inside of an obstacle.

When considering a nonholonomic systems there are a few problems which DRRRT does not address. In DRRRT much of the running time is spent in neighborhood finding.

When a new sample is chosen the nearest neighbor in the tree must be selected so that the tree can extend from the neighbor to the new sample. This results in a search over the entire tree. To limit this we introduce a topological bucketing neighborhood finder that limits this search to a smaller set of candidates associated with the embedding graph. Another issue pertains to the dynamics introduced by a nonholonomic problem. In a nonholonomic problem configurations may also consist of velocity parameters. The simple approach is to generate these velocities randomly. We introduce a method to bias the randomly generated velocity along the embedding graph provided by DRRRT to improve exploration. Lastly, we introduce a region weighting scheme. When generating a new sample, DRRRT must select a region to sample from. Previously, this was done uniformly over all regions and the environment. We aim to improve this by assigning each region a probability of being selected depending on the region's sampling history. If a region generates more successful samples its probability of being selected for future samples will increase [1]

We demonstrate the method as it applies to car-like robots. To do this we performed experiments in two simple environments with a car-like robot. The method is compared to the old DRRRT, SyCLoP, another workspace planner, and standard kinodynamic RRT in a uniform grid and a small maze.

[1] This work is done in collaboration with Andrew Bregger. The methods are the same between our theses. The differences are in our applications and results. We apply these methods to car-like robots in this paper, while his thesis focusing on applications to drones.

2. RELATED WORK

In this chapter, we discuss the important background information for the motion planning problem and other work related to this method.

2.1 Motion Planning

Motion planning is the task of finding a path through some environment from a start to a goal position. Traditionally, this path must fit constraints such as avoiding the obstacles and boundaries of the environment and allowing the object or robot to move along it without collision. In this paper, we discuss motion planning for holonomic and non-holonomic robots. A holonomic robot is a robot where all of its degrees of freedom (DOFs) are controllable. The DOFs of a robot parameterize its position and orientation. They include the robot's position, rotation, and joint angles if applicable. A non-holonomic robot is one where not all DOFs are controllable, such as a car, which cannot move laterally without first turning. The motion planning problem is often represented by the workspace and configuration space or $\mathcal{C}_{space}$.

The workspace of a motion planning problem is the environment which consists of obstacles and a boundary. $\mathcal{C}_{space}$ is the set of all configurations of a given robot. A configuration is one unique set of values for a robot's DOFs. For a simple robot in a 2-d world one configuration could be $q = \langle x, y, \theta \rangle$ where x and y are the robot's position in the world and θ is its rotation angle. $\mathcal{C}_{space}$ also consists of two subsets, free space ($\mathcal{C}_{free}$) and obstacle space ($\mathcal{C}_{obst}$). $\mathcal{C}_{obst}$ is the set of configurations in $\mathcal{C}_{space}$ that are in collision with an obstacle in the workspace and $\mathcal{C}_{free}$ is the set of configurations in $\mathcal{C}_{space}$ that are not in collision. With this information we can represent the motion planning problem as finding a continuous path of configurations in $\mathcal{C}_{free}$ from the start to goal configurations.

2.2 Sampling-based Planning

One common and effective technique for addressing the motion planning problem is sampling-based planning. The goal of sampling-based planning is to construct a graph that represents $\mathcal{C}_{free}$ by generating sample configurations in $\mathcal{C}_{free}$. These samples are then connected to form a graph or roadmap. Once the roadmap is constructed, the start and goal configurations can be connected to the closest point on the roadmap and a path can be found. One example of sampling-based planning is the Rapidly-exploring Random Tree [1], which is further explained in the next section.

2.3 Rapidly-exploring Random Trees (RRTs)

Rapidly-exploring Random Trees are a type of sampling based planning algorithm that are effective single query motion planning problems. That is problems consisting of only one start and one goal configuration. Rapidly-exploring Random Tree solves a problem by iteratively expanding outwards from root configuration (q_{root}). For each iteration, a random configuration ($q_{rand} \in \mathcal{C}_{space}$) is generated. Then the nearest configuration to q_{rand} in the tree (q_{near}) is found and is extended from q_{near} in the direction of qrand some distance Δd. The end position of the extension becomes a new configuration (q_{new}) which is added to the tree if and only if there is a valid path between q_{near} and q_{new}.

One specific type of RRT is Reachability-guided RRT (RGRRT) [4]. A reachable set is defined as a set of configuration that can be reached by a robot given its controls and configuration. A control is a force that can be applied to a robot to move it from one configuration to another. RGRRT uses the reachable set to bias the sampling. When generating samples, if q_{rand} is closer to q_{near} than any configuration in the reachable set q_{rand} is discarded. This approach allows to the RRT to better sample the unexplored space of the environment.

(a) Example embedding graph (b) Execution of Dynamic Region-biased RRT.

Figure 2.1: The embedding graph (a) represents paths of exploration through the workspace. These are explored by dynamic sampling regions that guide RRT growth (b).

2.4 Dynamic Region-biased RRT

In this paper, we extend Dynamic Region-biased RRT (DRRRT) [3] to better support kinodynamic motion planning problems. Dynamic Region-biased RRT is based on RRT with some key differences. DRRRT computes a representation of the workspace topology and uses dynamic sampling regions to guide an underlying RRT planner. The representation of the workspace topology is known as an embedding graph. The embedding graph is a skeleton of the workspace represented by an undirected graph. In this paper we use embedding graph, workspace skeleton, and skeleton interchangeably. Next the embedding graph is converted to a directed graph from the start to goal configurations called a flow graph. This represents the exploration direction of the robot from the start to goal. A region is a bounded volume in the workspace such as a bounding box or a bounding sphere. After computing the flow graph, a region is created at the beginning of the graph and is guided along the graph to the goal. If the flow graph splits, representing different paths, or homotopies, through the workspace, multiple regions are dynamically created to explore each path. At each iteration of the algorithm a region or the environment is chosen and a sample is generated within that region or environment. Finally, the underlying RRT extends to this new sample if it is valid (Figure 2.1).

2.5 Synergistic Comination of Layers of Planning

Synergistic Combination of Layers of Planning [5] (SyCLoP) addresses the problem of nonholonomic planning. In SyCLoP, the workspace is decomposed to construct a model of the problem. At each iteration of the algorithm a high-level planner searches this model for a feasible path which can be used to guide an underlying tree structure. They test their method on robots with high-dimensional dynamics including a unicycle, a flying unicycle, and a tractor trailer in environments with multiple narrow passages and a maze.

3. METHOD DESCRIPTION

3.1 Topological Bucketing

Algorithm 1 Algorithms for tree extension with topological bucketing.

```
function ExtendWithRegion(Region r, Tree t)
    q_rand ← Sample(r) // Or BiasedSample...
    candidates ← FindCandidates(r)
    q_near ← FindNearestNeighbor(candidates)
    q_new ← Extend(t, q_rand, q_near)
    if q_new ∈ C_free then
        BucketMap[r.Center()].Append(q_new)
    end if
end function
function FindCandidates(Region r)
    p ← r.GetCenter()
    candidates ← BucketMap[p]
    e ← r.GetSkeletonEdge()
    d ← 0
    while d < threshold do
        d += distance(p, e.PointBefore(p))
        p ← e.PointBefore(p)
        candidates.Append(BucketMap[p])
    end while
    return candidates
end function
```

One bottleneck in Dynamic Region-biased RRT is in neighborhood finding. This is caused by using a brute force method which searches the entire tree for the nearest configuration. To improve on this approach we would like to utilize the information provided by the embedding graph to limit the candidates for neighborhood finding. The solution to this is topological bucketing. The algorithm for this method is given in Algorithm 1.

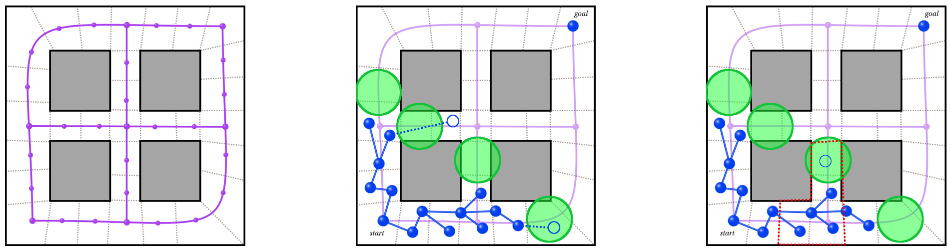
The embedding graph structure consists of vertices with edges connecting them. These edges have various edge points or intermediates along it, on which the region is centered (Magenta line and points in Figure 3.2a). When generating a sample we add the sample to a 'bucket' (Defined by blue lines in Figure 3.1) associated with the edge point at the center of the region. In doing this each successful sample is mapped to its nearest edge point. When finding the nearest neighbor, instead of searching the entire tree, we can use the buckets as input to the neighborhood finder and effectively reduce the size of the input.

Algorithm 1 explains this process. In Algorithm 1:1-9 a random sample is first generated and its candidates are found. Then these candidates are used as input to a brute force neighborhood finder. A standard RRT extend is then called from the random sample to the new configuration returned by the neighborhood finder. In Algorithm 1:6-8 the new configuration is added to the bucket associated with the current region's center. Here the bucket map is an associative container that associates a region's center point with a set of configurations or a bucket (Figure 3.2b). Finding the candidates of a random sample is done in Algorithm 1:10-21.

In order to determine which buckets to search over we initially set the candidates (Red outline in Figure 3.1c) to be the bucket associated with the current region's center (Algorithm 1:11-12). Then we traverse the embedding graph backwards for a distance $d < threshold$, adding the configurations in each bucket to the candidates set (Algorithm 1:15-19). For our purposes we set $threshold$ to be the maximum distance that the extender can extend.

In using this approach we observe two advantages over the standard brute force search method. First, the size of the input that the neighborhood finder must search over is reduced from the entire tree to a small portion of the tree stored in the nearest buckets. Second, the configurations in the candidates set are more likely to be near the newly sampled configuration as they come from the buckets which are at most a distance $d < threshold$

away from the sampled configuration.



(a) Embedded graph defines buckets

(b) RRT Growth

(c) Buckets used for neighborhood finding (red dotted lines)

Figure 3.1: Example progression of Bucket Neighborhood finder: (a) Embedded graph (magenta) and buckets (dotted lines); (b) RRT growth (c) Construction of candidate set (red outline) from local buckets.

3.2 Velocity Sampling

In kinodynamic motion planning each configuration can have DOF values that represent more than position, such as, velocity. When generating a random configuration one approach for giving it velocity is to generate a random linear velocity for the configuration. Although this method is fast, it can often lead to configurations which can only travel in an unhelpful direction. For example, it is possible for a velocity to be generated which directs a configuration backwards into the tree instead of towards unexplored free space (Yellow arrow in Figure 3.2a). This is another problem we can address using the information provided by the embedding graph.

The embedding graph represents the workspace topology and provides us with a guide from the start to the goal. The embedding graph also consists of multiple intermediate points along each edge which can be used to represent the direction of the graph (Red arrow in Figure 3.2a). These directions can be used to bias randomly generated velocities

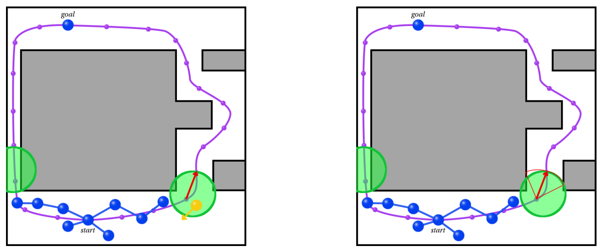
Algorithm 2 Algorithm for biasing velocity along skeleton.

```
1:  function BIASEDSAMPLE(Region r)
2:      q_rand ← Sample(r)
3:      e ← r.GetSkeletonEdge()
4:      p ← r.GetCenter()
5:      dir ← unit(e.PointAfter(p) − p)
6:      coeff ← unit(q_rand.LinearVelocity()) · dir
7:      if coeff < alpha then
8:          while coeff < alpha do
9:              q_rand ← Sample(r)
10:             coeff ← unit(q_rand.LinearVelocity()) · dir
11:         end while
12:     end if
13:     return q_rand
14: end function
```

to be 'along' the embedding graph. This approach is shown in Algorithm 2.

First, in Algorithm 2:2-4 a random configuration is generated from the current region with a random velocity. Then, from the region we obtain the current skeleton edge and point. With this information we can compute the direction of the skeleton, dir, as the unit vector between the current point and the next point on the skeleton. Next, we set $coeff$ to be the dot product between the configuration's unit linear velocity and the direction of the skeleton. In Algorithm 2:7-12 the goal is to minimize the difference between the configurations velocity and the skeleton direction by using the properties of the dot product. If the two velocities are in opposite directions then the dot product will return -1, provided the vectors are unit vectors. If the two velocities are parallel the dot product will return 1. We use a parameter, α, to maximize the dot product of the two directions and generate a velocity which is along the skeleton within some bounds. This is shown in Algorithm 2:8-9 where a new configuration and velocity is generated until the dot product between the velocity and the skeleton direction becomes larger than α and acceptable to use.

Since generating random samples is a relatively fast operation it is acceptable to repeatedly sample in this manner. Additionally, α can be tuned to increase the likelihood to generate an acceptable velocity and decrease the total number of additional samples needed to find an acceptable velocity (red outline in Figure 3.2b).



(a) (b)

Figure 3.2: Motivation for Velocity Biasing: (a) General direction of embedded graph (red arrow) and the direction of new configuration (yellow arrow). (b) Allowable direction for velocities (red cone).

3.3 Region Weighting

At each iteration of Dynamic Region-biased RRT a region is chosen and a new configuration is generated from that region. Originally, this decision was made uniformly over all the regions and the entire environment itself. That is, each region, including the entire environment, had an equal chance of being chosen for sampling. We know that for the most part we want samples to be generated in a region, not the environment. Additionally, we want to choose regions which have a history of generating successful samples that help guide the RRT. To accomplish this we use a new region weighting scheme which computes a probability, p_i, for each region $\langle r_0, r_1, \ldots, r_n \rangle$ and the environment. We also define a weight for each region $w_i = s/t$, where s is the number of successful samples generated in region i and t is the total samples generated in region i. The probability is defined to be:

$$p_i = (1 - \gamma)\frac{w_i}{\sum_{j=1} K w_j} + \gamma\frac{1}{K+1} \tag{3.1}$$

where gamma is a constant in the range $[0, 1]$ and K is the total number of current regions. The first term is determined by the ratio of the region's weight to the sum of all current regions' weights. This allows us to determine, to some extent, how well this region is performing. The second term represents uniform probability to select a region based on the input parameter γ. Here $K + 1$ is used to include the environment. If $\gamma = 1$ then the probability is exactly uniform, and if $\gamma = 0$ the probability is strictly based on the region's weight compared to the sum of all regions' weights. Since this probability is based on the weight of all current regions, we must dynamically update each region's probability when any region is added, deleted, or generates a sample.

Using this scheme we effectively bias sampling to regions which historically generate more successful samples, and thus, are more likely to be in areas of free space which have higher clearance between obstacles and more space for exploration.

4. EXPERIEMENTS AND RESULTS

4.1 Experiments

All methods were implemented in a C++ motion planning library (PMPL) developed in the Parasol Lab at Texas A&M University. PMPL uses a distributed graph data structure from the Standard Template Adaptive Parallel Library (STAPL) [6], a C++ library for parallel computing developed at Parasol Lab.

All experiments were performed on a desktop, at Parasol Lab, running CentOS 7 with Intel® Core™ i7-3770 at 3.40 GHz, 16 GB of RAM, and the GNU gcc compiler version 4.8.5.

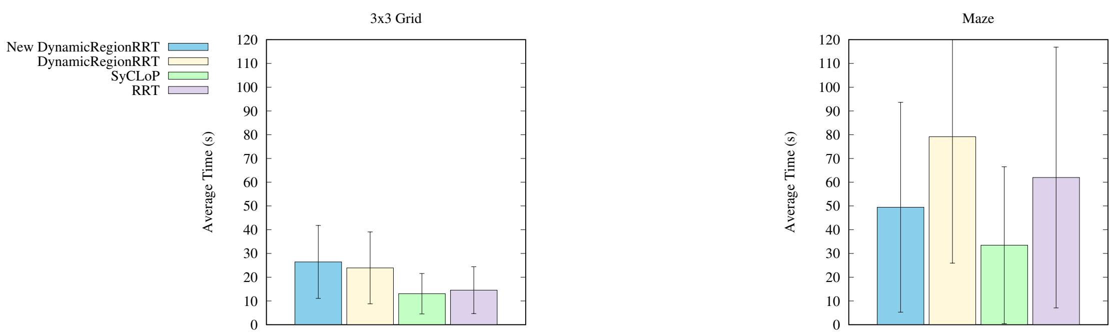
4.2 Results

The new Dynamic Region-biased RRT was compared against the original Dynamic Region-biased RRT [3], SyCLoP [5], and a standard kinodynamic RRT. SyCLoP is chosen for comparison because it is another example of a planner which utilizes workspace information during planning time. The new algorithm is demonstrated on a nonholonomic robot (car) in two environments, a 3 by 3 grid used to represent a grid-like city where a vehicle would need to make sharp turns, and a maze which consists of non-uniform paths and frequent turning. These environments are shown in Figure 4.1. The car-like robot used in these environments is a 6 DOF rigid body using a control set which allows forward, backward and rotational movement. This robot does not exactly simulate the dynamics of a car. For our purposes the constraints needed to simulate a car, such as lateral movement and not being able to rotate without moving forward or backwards, are not implemented. These additions are left to future work. Each time an extension is made the best control is selected and applied with a fixed timestep.

Each experiment trial ran until the query was solved (success) or the trial reached the

(a) 3x3 Grid (b) Maze

Figure 4.1: The experiment environments shown with the query for the robot (car). The start configurations are shown in red and goals in blue.



(a) 3x3 Grid (b) Maze

Figure 4.2: On-line planning time comparing the new Dynamic Region-biased RRT with the original Dynamic Region-biased RRT SyCLoP, and RRT in two nonholonomic problems. The time values (seconds) are an average over 18 trials. The error bars indicate standard deviation of the times.

node (20k) or time limit. The time limit was set to 3 minutes for the 3x3 grid and maze. A total of 20 experiments were performed for each method on each environment. After trimming the best and worst performance form the 20 runs we are left with 18 runs shown in Figure 4.2 with their average total runtimes and error bars representing the standard deviation of the runtimes. Additionally, success rates for each method and environment are shown in Table 4.1.

4.2.1 Discussion

As shown in Figure 4.2a, in the 3x3 grid environment the times are similar for each experiment, but SyCLoP proves to be slightly faster. The new additions to DRRRT do not appear improve the performance over the original DRRRT in this environment. Due to the simplicity of this problem RRT performs very well. The methods can solve this grid by navigating around the edge and potentially only making one turn on the outer corner. This explains why the times are faster on this environment when compared to maze. One way to increase the difficulty would be to remove the space beside the obstacles on the left and right side of the grid. This would force the planner to make more turns through the obstacles. Experiments in more difficult environments are left to future work.

In maze DRRRT performs well compared to the original DRRRT and RRT. In this environment there is one feasible path and our method finds it immediately due to pruning of the embedding graph. Branches of the graph that lead to dead-ends in the maze will be pruned and planning will not be guided in that direction.

The error bars in maze are high for all methods indicating some inconsistency involved when solving this problem. These inconsistencies may be caused by the reaching unrecoverable states. A configuration is known as an unrecoverable state if it is difficult to make a successful extension from it. If these occur extending the tree can become very difficult and may lead to failure to solve the problem. For DRRRT we intend to address this by introducing reachability guidance. This is discussed in Section 5.1

Another possible cause for higher runtimes in 3x3 grid is choosing to explore a path which is not optimal. In this environment, there are many paths to take to the goal, some of which could take longer than others due to the extra turning necessary. If these paths become the priority for sampling then the overall performance can be slower.

In the maze, the extra paths are trimmed reducing this problem, however the tight turns

Table 4.1: Success rates in each experiment.

Environment	New Dynamic Region-biased RRT	Dynamic Region-biased RRT	SyCLoP	RRT
3x3 Grid	100%	100%	100%	100%
Maze	94%	100%	100%	100%

cause slower times overall.

These results show us that our initial implementation may not be correct and there areas for improvement in our algorithm. This leads to our discussion on future work, which includes improving our the presented methods and introducing a few new ideas.

5. CONCLUSION

In this paper, we introduced three algorithmic additions to Dynamic Region-biased RRT, a topological bucketing approach for neighborhood finding, a biased method for sampling velocities, and a weighting scheme for choosing regions. We attempt to show how these changes are applicable to nonholonomic problems, but the results show that there is room for more improvements. We discuss how we plan to make these improvements in the next section.

5.1 Future Work

In the future, we will investigate the causes of the poor running times. Specifically, bucketing improves neighborhood finding times, however it appears to have an adverse effect on the overall runtime. Another area for improvement is in velocity biasing. In addition to biasing the direction of a configuration's velocity we would like to dynamically adapt the velocity to the current speed of the robot and the expected extenstion distance. This extension distance can also be dynamically updated based on the speed of the robot and the size and direction of local embedding graph edges. Currently the extension distance is constant for each environment. However, many environments (especially cluttered spaces) can have different regions of the environment which would need different extension distances. For example, an environment could have one region where the free space is large and open, but another region with a narrow passage. In the former case a larger extension distance would allow the robot to explore this more open space quickly, while a short extension distance would allow more turning to navigate tighter spaces in the latter case.

Additionally, we would like to introduce reachability guidance to Dynamic Region-biased RRT. In reachability guidance, the reachable set of a configuration is considered when making extensions. A reachable set is the set of configurations that a given configuration can reach by applying all provided controls. We will work on methods to compute or approximate the reachable set in order to avoid extending to configurations which have poor reachability.

Lastly, we want to further test this method on more interesting environments. For a car-like robot this could be a cluttered environment (unlike the grids used in this paper) or a city street layout which is not uniform. We could also test the car with a trailor attached, which would introduce more complicated dynamics.

REFERENCES

[1] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” tech. rep., 1998.

[2] S. M. LaValle and J. J. Kuffener Jr, “Randomized kinodyanamic planning,” 2001.

[3] J. Denny, R. Sandstrom, A. Bregger, and N. M. Amato, “Dynamic region-biased rapidly-exploring random tree (wafr 2016),” 2016.

[4] A. Shkolnik, M. Walter, and R. Tedrake, “Reachability-guided sampling for planning under differential constraints,” pp. 2859–2865, 2009.

[5] E. Plaku, L. E. Kavraki, and M. Y. Vardi, “Motion planning with dynamics by a synergistic combination of layers of planning,” *IEEE Transactions on Robotics*, vol. 26, pp. 469 – 482, 2010 2010.

[6] P. An, A. Jula, S. Rus, S. Saunders, T. Smith, G. Tanase, N. Thomas, N. Amato, and L. Rauchwerger, “STAPL: A standard template adaptive parallel C++ library,” Jul 2001.